**What are Servlets?**

Servlet is a technology i.e. used to create web application.

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Servlet is a web component that is deployed on the server to create dynamic web page.

Servlet is an API that provides many interfaces and classes including documentations.

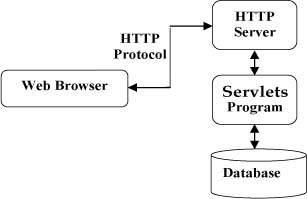
Servlet is an interface that must be implemented for creating any servlet.

Servlet is a class that extends the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.

Servlet is a java file which can take the request from client on the internet and it can process the request and provide the response in the format of HTML page.

**Servlets Architecture:**

Following diagram shows the position of Servlets in a Web Application.



**Servlets - Life Cycle**

A servlet life cycle can be defined as the entire process from its creation till the destruction. There are manly three stages in servlet life cycle

|  |
| --- |
| Servlet is in service  init() destroy()  Servlet is created when the work of servlet is done  Service() the last stage is destroy.  (here loop because servlet can be  invoked many times and the  method (i.e serice()) is in this stage is called  Service stage. It is the most important  Stage in any servlet life cycle.) |

• The servlet is initialized by calling the init () method.

• The servlet calls service() method to process a client's request.

• The servlet is terminated by calling the destroy() method.

**The init() method :**

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

public void init() throws ServletException {

// Initialization code...}

**The service() method :**

The service() method is the main method to perform the actual task. Whenever request is made from client(browser) the servlet container (i.e. web server) calls the service() method to handle requests and response back to the client.

The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

public void service(ServletRequest request,

ServletResponse response)

throws ServletException, IOException{

}

The service () method is called by the container and service method invokes doGet, doPost methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods within each service request.

Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

**The destroy() method :**

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

public void destroy() {

// Finalization code...

}

**Java Servlet**

**Six Steps to Running Your First Servlet**

1. Create a directory structure under Tomcat for your application.
2. Write the servlet source code. You need to import the javax.servlet package and the javax.servlet.http package in your source file.
3. Compile your source code.
4. Create a deployment descriptor.
5. Run Tomcat.
6. Call your servlet from a web browser.

**Step 1: Create a Directory Structure under Tomcat**

When you install Tomcat, several subdirectories are automatically created under the Tomcat home directory. One of the subdirectories is webapps. The webapps directory is where you store your web applications. A web application is a collection of servlets and other contents installed under a specific subset of the server's URL namespace. A separate directory is dedicated for each servlet application. Therefore, the first thing to do when you build a servlet application is create an application directory.

Let’s see how to create a directory structure for an application called myApp.

|  |
| --- |
| classes  WEB-INF  myApp  Webapps  Web.xml TstingServlet.class |

Create a directory called myApp under the webapps directory. The directory name is important because this also appears in the URL to your servlet.

Create the src and WEB-INF directories under myApp, and create a directory named classes under WEB-INF. The src directory is for your source files, and the classes directory under WEB-INF is for your Java classes. If you have html files, you put them directly in the myApp directory. You also may want to create a directory called images under myApp for all your image files.

**Step 2: Write the Servlet Source Code**

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

import java.util.\*;

public class TestingServlet extends HttpServlet {

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

PrintWriter out = response.getWriter();

out.println("<HTML>");

out.println("<HEAD>");

out.println("<TITLE>Servlet Testing</TITLE>");

out.println("</HEAD>");

out.println("<BODY>");

out.println("Welcome to the Servlet Testing Center");

out.println("</BODY>");

out.println("</HTML>");

}

}

You can save the source files anywhere in your computer;

**Step 3: Compile Your Source Code**

Note:

Like any other Java program, you need to compile a servlet by using the Java compiler “**javac”** and after compilation the servlet application, it would be deployed in a configured environment to test and run.

Set “**classpath”** variable with value as follows

|  |
| --- |
| C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\servlet-api.jar; |

**Step 4: Create the Deployment Descriptor**

A deployment descriptor is an optional component in a servlet application, taking the form of an XML document called **web.xml**. The descriptor must be located in the WEB-INF directory of the servlet application.

For this step, you now need to create a web.xml file and place it under the WEB-INF directory under myApp.

The web.xml for this example application must have the following content.<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name> TestingServlet </servlet-name>

<servlet-class>TestingServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name> TestingServlet </servlet-name>

<url-pattern>/ TestingServlet </url-pattern>

</servlet-mapping>

</web-app>

The web.xml file has one element: web-app. You should write all your servlets under <web-app>. For each servlet, you have a <servlet> and <servlet-mapping> element.

you need the <servlet-name> and <servlet-class> elements. The <servlet-name> is the name for your servlet, by which it is known to Tomcat. The <servlet-class> is the compiled file of your servlet without the .class extension.

Having more than one servlet in an application is common. For every servlet, you need a <servlet> element in the web.xml file. For example, the following code shows how the web.xml looks if you add another servlet called Login.

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name> TestingServlet </servlet-name>

<servlet-class>TestingServlet</servlet-class>

</servlet>

<servlet>

<servlet-name>Login</servlet-name>

<servlet-class>Login</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name> TestingServlet </servlet-name>

<url-pattern>/ TestingServlet </url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name> Login </servlet-name>

<url-pattern>/ Login </url-pattern>

</servlet-mapping>

</web-app>

**Step 5: Run Tomcat**

If it is not already running, you need to start Tomcat.

**Step 6: Call Your Servlet from a Web Browser**

You are ready to call your servlet from a web browser. By default, Tomcat runs on port 8080 in **myApp** virtual directory under the servlet subdirectory. The servlet that you just wrote is named Testing. The URL for that servlet has the following format:

|  |
| --- |
| [**http://domain-name/virtual-directory/servlet/servlet-name**](http://domain-name/virtual-directory/servlet/servlet-name)  **eg: http://localhost:8080/myApp/Testing.** |

**Servlets - Form Data**

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GET method:

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows:

http://www.test.com/hello?key1=value1&key2=value2

The GET method is the defualt method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limtation: only 1024 characters can be in a request string.

This information is passed using QUERY\_STRING header and will be accessible through QUERY\_STRING environment variable and Servlet handles this type of requests using doGet() method.

**POST method:**

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using doPost() method.

**Reading Form Data using Servlet:**

Servlets handles form data parsing automatically using the following methods depending on the situation:

• getParameter(): You call request.getParameter() method to get the value of a form parameter.

**GET Method Example Using URL:**

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first\_name=Ankit&last\_name=Poddar

Below is HelloForm.java servlet program to handle input given by web browser. We are going to use getParameter() method which makes it very easy to access passed information:

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class HelloForm extends HttpServlet {

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Using GET Method to Read Form Data";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " +

"transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor=\"#f0f0f0\">\n" +

"<h1 align=\"center\">" + title + "</h1>\n" +

"<ul>\n" +

" <li><b>First Name</b>: "

+ request.getParameter("first\_name") + "\n" +

" <li><b>Last Name</b>: "

+ request.getParameter("last\_name") + "\n" +

"</ul>\n" +

"</body></html>");

}}

Assuming your environment is setup properly, compile HelloForm.java as follows:

> javac HelloForm.java

If everything goes fine, above compilation would produce HelloForm.class file. Next you would have to copy this class file in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes and create following entries in web.xml file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

<servlet>

<servlet-name>HelloForm</servlet-name>

<servlet-class>HelloForm</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloForm</servlet-name>

<url-pattern>/HelloForm</url-pattern>

</servlet-mapping>

Now type http://localhost:8080/HelloForm?first\_name=Ankit &last\_name=Poddar in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result:

GET Method Example Using Form:

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this imput.

<html>

<body>

<form action="HelloForm" method="GET">

First Name: <input type="text" name="first\_name">

<br />

Last Name: <input type="text" name="last\_name" />

<input type="submit" value="Submit" />

</form>

</body>

</html>

Keep this HTML in a file Hello.htm and put it in <Tomcat-installation-directory>/webapps/ROOT directory. When you would access http://localhost:8080/Hello.htm, here is the actual output of the above form.

First Name: Last Name:

Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

POST Method Example Using Form:

Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is HelloForm.java servlet program to handle input given by web browser using GET or POST methods.

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class HelloForm extends HttpServlet {

// Method to handle GET method request.

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Using GET Method to Read Form Data";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " +

"transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor=\"#f0f0f0\">\n" +

"<h1 align=\"center\">" + title + "</h1>\n" +

"<ul>\n" +

" <li><b>First Name</b>: "

+ request.getParameter("first\_name") + "\n" +

" <li><b>Last Name</b>: "

+ request.getParameter("last\_name") + "\n" +

"</ul>\n" +

"</body></html>");

}

// Method to handle POST method request.

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

Now compile, deploy the above Servlet and test it using Hello.htm with the POST method as follows:

<html>

<body>

<form action="HelloForm" method="POST">

First Name: <input type="text" name="first\_name">

<br />

Last Name: <input type="text" name="last\_name" />

<input type="submit" value="Submit" />

</form>

</body>

</html>